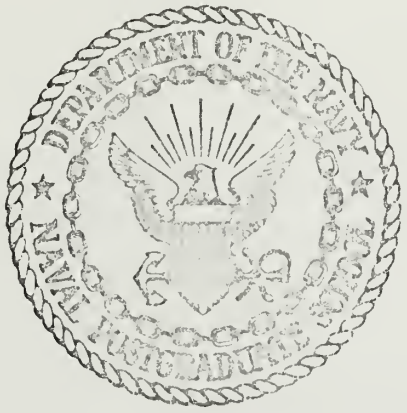
AN INTERACTIVE GRAPHICAL DEBUGGING SYSTEM

Allan Warren Walker

United States
Naval Postgraduate School



THESIS

AN INTERACTIVE GRAPHICAL DEBUGGING SYSTEM

Allan Warren Walker

Thesis Advisor E. A. Singer

June 1971

Approved for public release; distribution unlimited.

T139857

An Interactive Graphical Debugging System

by

Allan Warren Walker
Ensign, Supply Corps, United States Navy
B.A., University of Washington, 1968

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the
NAVAL POSTGRADUATE SCHOOL
June 1971

W 2204
c.1

ABSTRACT

A system is described which provides an interactive graphical debugging facility for user programs. This system is implemented on an Adage AGT-10 and is operational for online debugging of higher-level language programs executing on an XDS 9300 host computer. System architecture and implementation are discussed. A formal definition of the DEBUG Command Language is given and a description of the utilization of the commands for program debugging is presented.

TABLE OF CONTENTS

I. INTRODUCTION ---------- 7

A. DEBUGGING SYSTEMS ---------- 7

B. EXECUTION-TIME DEBUGGING ---------- 8

C. ONLINE VERSUS OFFLINE DEBUGGING ---------- 9

II. AN INTERACTIVE GRAPHICAL DEBUGGING SYSTEM ---- 11

A. NON-INTERFERENCE LEVEL ---------- 12

B. INTERACTIVE LEVEL ---------- 12

C. AUTOMATIC SYMBOL LEVEL ---------- 13

III. SYSTEM ORGANIZATION ---------- 14

A. MAIN COMMUTATOR ---------- 14

B. COMMAND PROCESSING ---------- 15

C. INPUT-OUTPUT ---------- 16

IV. DEBUG COMMAND LANGUAGE ---------- 18

A. DEFINITIONS ---------- 18

1. Symbolic Reference ---------- 18

a. Command Names ---------- 18

b. Symbol ---------- 18

2. Command ---------- 19

a. Action Command ---------- 19

b. Symbol Definition Command ---------- 19

c. Value Inquiry Command ---------- 19

3. Operand List---------- 20

4. Operands---------- 20

a. Character String Constant---------- 20

b. Octal Constant---------- 20

c. Decimal Constant---------- 21

d. Floating Point Constant---------- 21

e. Symbol---------- 21

f. Expression---------- 21

B. COMMAND LANGUAGE FORMAT---------- 22

1. Small Letters---------- 22

2. Capital Letters---------- 22

3. Square Brackets---------- 22

4. Braces---------- 22

V. SYSTEM COMMAND CAPABILITIES---------- 24

A. NON-INTERFERENCE LEVEL---------- 24

1. DUMP Command---------- 24

2. UNDUMP Command---------- 25

3. HOLD Command---------- 25

4. FREE Command---------- 25

5. PAGE Command---------- 25

6. BACK Command---------- 25

7. LIST Command---------- 26

8. UNLIST Command---------- 26

9. GATED Command---------- 26

10. Symbol Definition Command---------- 26

11. Value Inquiry Command---------- 27

12. DELETE Command---------- 27

13. AMRMX Command---------- 27

B. INTERACTIVE LEVEL---------- 27

1. FILL Command---------- 27

2. BABY Command---------- 28

3. TRAP Command---------- 28

4. UNTRAP Command---------- 28

5. ATEX Command---------- 28

6. STOP Command---------- 29

7. GO Command---------- 29

8. SLOW Command---------- 29

9. FAST Command---------- 29

C. AUTOMATIC SYMBOL LEVEL---------- 29

1. SYMBOLS Command---------- 30

VI. INTERNAL SYSTEM STRUCTURE---------- 31

A. INTERRUPT HANDLING CONSIDERATIONS---------- 31

B. COMMAND PROCESSING---------- 32

1. Command Decoder---------- 32

2. Command Processors---------- 33

C. SYMBOL TABLE ORGANIZATION---------- 34

D. XDS 9300 COMMUNICATION---------- 35

1. Non-Interactive Communication---------- 35

2. Interactive Communication---------- 36

VII. USING THE DEBUGGING SYSTEM---------- 37

VIII. OBSERVATIONS AND CONCLUSIONS---------- 41

A. SUGGESTIONS FOR FURTHER DEVELOPMENT----- 42

B. CONCLUDING REMARKS---------- 44

APPENDIX A NON-INTERFERENCE COMMANDS---------- 45

APPENDIX B INTERACTIVE COMMANDS---------- 54

INITIAL DISTRIBUTION LIST---------- 61

FORM DD 1473---------- 62

I. INTRODUCTION

Frequently one of the most time and money consuming efforts in the development of computer programs and their revisions is that devoted to debugging. In the present context debugging refers to the detection, location and elimination of program bugs which may have originated as errors in program logic or clerical mistakes such as in keypunching. Typically, with a batch processing computer configuration, the debugging effort will involve repeated computer runs, hand simulation of portions of the program, and not infrequently, the insertion of numerous WRITE or PRINT statements solely to provide an indication of program flow and/or values attained by program variables in the course of execution. When the bugs have been eliminated these additional statements are removed to provide the finished program.

A. DEBUGGING SYSTEMS

The programmer might be greatly aided in reducing the necessary effort of program checkout if a powerful debugging system were available to assist him. Such a system should enable a user to observe pertinent aspects of the execution of his program without requiring modifications to it. Most manufacturers of computer systems provide at least a rudimentary debugging package with their operating systems. Unfortunately, some of these are so complex in their operation as to preclude all but the most experienced programmers from using them.

Nearly all such packages require that the user prepare for debugging prior to compilation and execution of his program. Often a special compilation mode is necessary and changes in debugging requests cannot be made without reloading and reexecution of the program.

Basically, debugging systems may be divided into categories depending upon when they interact with the user's program, i.e. compile-time, load-time or execution-time debugging. Although there are certain advantages to the first two of these methods, they necessitate a fairly complicated debugging package that is essentially built into the operating system, and is restricted to handling only programs from a particular set of language processors. It is the intent of this paper to be concerned solely with execution-time debugging, which is readily adaptable to the system under consideration.

B. EXECUTION-TIME DEBUGGING

Execution-time debugging may be language independent to a greater degree than the other two types mentioned above, and may be more readily adapted to a computer without extensive modifications to the operating system. However, in some implementations it often requires that the user be highly versed as a machine language programmer, or in fact as a machine operator, to make use of its features. On frequent occasions a user who is well acquainted with the internal operation of a particular computer may choose to debug portions of a program by sitting at the computer console and single-stepping through portions of

his program, displaying contents of registers, etc. Although this may be applicable in many situations, in practice it is evident that computer control consoles are not well adapted to debugging. Some means of providing a console facility more suitable to debugging might be desirable.

It has been mentioned that one of the drawbacks of execution-time debugging is that the user might need to be competent as a machine language programmer. This stems from the need to understand the significance of the contents of registers displayed in binary, octal, etc. and the lack of most information relating to such things as higher level language variable names at execution time. It will be shown here how the use of a small peripheral processor can reconstruct sufficient information to be of value to the user of a higher level language.

C. OFFLINE VERSUS ONLINE DEBUGGING

A further subdivision of most forms of debugging may be made, namely into online and offline systems. The distinction between the two is whether or not the user is able to interact with his program for debugging purposes during the course of its execution. It is often most desirable for the user to be able to debug his programs online, as from the author's experience the very nature of program debugging often makes it difficult to predict the areas of a program in which errors will occur. The debugging process is in reality an interaction between the user and his program, and the opportunity for the user to debug

online heightens this interaction. Additionally, the facility to enter temporary patches and fixes during execution as the errors are detected may prevent many nearly-duplicate program runs.

II. AN INTERACTIVE GRAPHICAL DEBUGGING SYSTEM

The system described here is an attempt to provide an online, execution-time debugging system in an environment where such a package was not formerly available. It includes the functions of an extended computer operating console specifically adapted for debugging with features that are useful to the FORTRAN user. Although an understanding of machine language is not necessary for use of all features, it is advantageous for the most complete utilization of DEBUG.

This debugging system (DEBUG) has been developed for use with the Xerox Data Systems 9300/Adage Graphics Terminal-10 system in the Electrical Engineering Department Hybrid Computer Laboratory at the Naval Postgraduate School. This machine configuration consists of one XDS 9300 central processing unit with input/output equipment and a high speed bi-directional data channel attached to two AGT-10 graphics terminals. In the normal mode of operation on the configuration a FORTRAN program on the XDS 9300 utilizes one of the AGT-10 terminals as a slave device for interactive graphical processing. Machine utilization includes faculty and student research, thesis projects and laboratory exercises, necessitating a considerable and continuing effort in individual program debugging.

DEBUG operates on either of the AGT-10 graphics terminals and is used to debug programs on the XDS 9300 which may be using the

other AGT-10 as a slave graphics device. The nature of the graphics interface between the XDS 9300 and the AGT-10 is such that the majority of information desired by the programmer is available in 9300 core memory. No alterations to the existing XDS 9300 Real Time Monitor were necessary. Three levels of interaction with the user program are available, their use being dependent upon the user's competence in machine language programming, the nature of the errors he is attempting to locate and the possible desire to avoid modification of the source program.

A. NON-INTERFERENCE LEVEL

At this level of interaction, the debugging system acts solely as an observer, providing dynamic core dumps, displaying values in locations, and interpreting such values. No modifications are made to the execution of the running program.

B. INTERACTIVE LEVEL

For most debugging operations, the user wishes to interact with his program, and may do so in DEBUG by inserting breakpoint traps, 'babysitting' on a location for a change in value, altering program code or data contents, or decreasing the execution speed of the program. This mode does not modify the user source program itself, but may alter the contents of the execution-time core memory image of the program at the specific request of the user.

C. AUTOMATIC SYMBOL LEVEL

If the programmer is willing to allow a reallocation of his program in 9300 core memory, the possibility of using a significantly greater amount of storage, and knows prior to compilation that he wishes to use the debugging package, he may utilize several additional debugging aids. The most convenient of these is to have all the symbols, such as FORTRAN variable names, predefined and automatically available to the user at execution time through DEBUG. This feature requires that the user include a 'blanket' NAMELIST statement in his FORTRAN program.

DEBUG uses the graphical display subsystem of the AGT-10, with a fourteen inch display area, for presenting all information requested by user debugging commands. Depending upon the type of commands issued by the user and the screen area currently occupied the debugging data may be accumulated on the display screen or will replace currently displayed information. Most displays are dynamic and are updated continually as the 9300 program execution progresses. Command input from the user is typed on the keyboard of a model 33 teletype-writer, with an image of the current input buffer displayed on the screen. Due to the utilization of a separate processor for debugging, with the ability to directly access XDS 9300 core memory, DEBUG makes no demands on the resources of the attached XDS 9300 computer, other than core memory access conflicts allowing program execution to proceed essentially unaffected except when specifically interrupted by the user.

III. SYSTEM ORGANIZATION

The Interactive Graphical Debugging System consists of a single program package for the AGT-10. This package is coded entirely in the Adage Extendable Program Translator (ADEPT) assembler language and consists of approximately 1800 statements. Once initiated, the program package operates without the assistance of any external subroutines or monitor functions. Although DEBUG does not communicate with the AGT monitor system, AMRMX, they are entirely compatible and DEBUG provides the facility to rapidly return control to the monitor under operator command. The DEBUG program package includes a main commutator, command processing routines, and an input-output section.

A. MAIN COMMUTATOR

The main commutator is the heart of the program package. It consists of a loop with a series of gates to various program segments. After initialization, the program loops continually through the commutator, with the exception of interrupts generated by the input-output hardware. Each gate in the commutator is a branchpoint to an individual program section, and each gate may be either open or shut at any particular time as needed to control the processing of operator requests. Most interrupts from input-output servicing cause a gate in the commutator to be opened, thus allowing the actual processing done at

interrupt time to be minimal. This technique also ensures that program code that cannot logically be executed out of sequence will not be improperly initiated by an interrupt.

B. COMMAND PROCESSING

When the input-output routines determine that the user has completed typing a request, which must end with the carriage return character, a gate in the commutator is opened to the command decoding program segment. This program segment scans the input text line, breaks it into individual arguments as delimiters are encountered and stores the arguments. Each argument is in turn examined, determined to be one of several types and properly decoded to a common format. At this point, all BCD arguments are checked for a match in the symbol table, and if found the equivalent value is substituted for the symbol. Simple arithmetic addition and subtraction of subfields within arguments is permitted, thus allowing the user to refer to such things as locations relative to a known symbol.

After all arguments have been processed, the command itself is looked up in a table of predefined commands and the appropriate program segment is executed. If the task to be performed is short or of a one-time nature it is executed immediately, otherwise a gate to perform it is set in the main commutator. After the task is performed or the gate set, the command processor regains control, closes its own gate and returns to the commutator. From this point forward, all

user initiated commands of a continuous nature will be performed by individual program segments tailored to the specific function.

The command processor includes various error messages for improper arguments and invalid commands, which reply to the operator via the teletypewriter printer and terminate processing of the current user command.

C. INPUT-OUTPUT

The input-output program segment consists of a set of subroutines to process interrupts from the input-output hardware, and to initiate requests for input or output. Included in this segment are routines for assembling or disassembling characters bitwise for the serial teletypewriter interface, transfering information to or from the XDS 9300 core memory, and driving the AGT graphics display subsystem. Subroutines for character set conversion, numeric value conversion and error message generation are also located in this segment.

Due to the nature of the debugging process, in which the user may make frequent requests of the system, the teletypewriter input subroutines provide facilities for ease of operation and correcting mistakes before the requests are executed by DEBUG. Backspace and line erase functions are provided by designated keys on the keyboard. The current content of the input typing buffer is continuously displayed near the lower edge of the display screen, so that the operator may verify his requests without looking away from the display. The interaction of the

command decoder with the input-output routines is such that the input text line is erased from the left as each argument is processed. If an error is detected by the command processor, the operator may readily determine the point at which it was detected by observing how much of the command has been erased when the error message occurs.

IV. DEBUG COMMAND LANGUAGE

All processing and displays generated by DEBUG are the direct result of user initiated requests in the DEBUG command language. Requests in the command language are always inputted through the AGT-10 teletypewriter keyboard, and may be entered at any time according to the desires of the user.

A. DEFINITIONS

Some definitions which are necessary for understanding and utilization of the command language are given here:

1. Symbolic Reference

A symbolic reference is a word having up to eight alphameric characters, at least one of which is non-numeric. They are used for two purposes:

a. Command Names

A command name is a name used to identify a particular command.

b. Symbol

A symbol is a name which is equated to an arithmetic or character string constant. Such symbols may be used as command operands interchangeably with the constants they represent. Symbols are defined individually by the use of the symbol definition command, or as a group in the automatic symbol definition mode. In the latter

case, the defined symbols are equated to the octal location value of corresponding program variable.

2. Command

A command is some specific instruction issued by the user which will cause a pre-defined sequence of operations to occur. Commands are of three classifications:

a. Action Command

An action command is a command which initiates some action desired by the user. These commands normally alter the display on the AGT-10 graphics unit, cause a given test condition to be enabled or alter the course of execution of the program being debugged. Action commands consist of a command name possibly followed by an operand list, depending upon command type and user intent.

b. Symbol Definition Command

A symbol definition command is a command which equates a symbol to a specified value. These commands cause no changes in the display or program execution when they are executed. Symbol definition commands consist of the symbol to be defined, the equals sign and a single operand prescribing the value to be attached to the symbol.

c. Value Inquiry Command

A value inquiry command is one which requests that the value of an operand or symbol be displayed. These commands cause no changes in the display or program execution other than the requested

value which is presented in the area of the input typing buffer display until the next command is typed. Value inquiry commands consist of a symbol or operand followed by the equals sign.

3. Operand List

Most commands require some modifying information. This information is supplied along with the command name in the operand list. An operand list is a series of one to four operands, separated by commas, with the entire list enclosed in left and right parentheses.

4. Operands

An operand is an element in an operand list, or may be the value to be equated in a symbol definition command. Operands may be any of the following forms:

a. Character String Constant

A character string constant is a series of up to eight alphameric characters, at least one of which must be non-numeric. It has the value of the octal equivalent of the BCD characters in the string. Embedded blanks are permissible. Character string constants of less than eight characters are filled with trailing blanks.

b. Octal Constant

An octal constant is an actual parameter consisting of up to eight octal digits. Octal digits may be any character from the set (0, 1, 2, 3, 4, 5, 6, 7). An octal constant with less than eight octal digits is internally filled with leading zeroes.

c. Decimal Constant

A decimal constant is an actual parameter consisting of up to six decimal digits followed by a decimal point. The maximum value of a decimal constant is 163839. Decimal digits may be any character from the set (0, 1, 2, 3, 4, 5, 6, 7, 8, 9). Decimal constants are internally filled with leading zeroes.

d. Floating Point Constant

A floating point constant is an actual parameter consisting of a decimal constant followed by a decimal fraction part, where the decimal fraction part consists of up to five decimal digits.

e. Symbol

A symbol used as an operand is replaced by the value which that symbol is defined to have. Note that symbols are of the same form as character string constants. Whenever an alphamerical operand is encountered, it is used as a symbol if such a symbol has been defined, otherwise it is taken as a character string constant.

f. Expression

An expression is any sequence of subfields joined by the arithmetic operators + or -. Each subfield may be any type of operand except an expression. The value of the expression is the result of performing the given arithmetic operations on the subfields from left to right. Two's complement arithmetic is used to maintain compatibility with the XDS 9300 internal representation of numbers. It is permissible for an expression to start with either an operator or a subfield, but it

must end with a subfield. Neither double operators nor double subfields are permitted.

B. COMMAND LANGUAGE FORMAT

Action commands will be given in the following format:

COMMAND	OPERANDS
CMD	(operand1[, {1 / 2}, operand3])
WHERE	
operand1 = ... etc.	

The following notational conventions will be used:

1. Small Letters

Small letters in the command description represent values, names, etc. that must be replaced by meaningful coding by the user. These are further explained in the WHERE section.

2. Capital Letters

Capital letters in the command description are to be coded as shown. The same applies to the following special characters:) (= ,

3. Square Brackets

[ ] indicate that the contents are optional and need only be coded if the default is not to be taken.

4. Braces

{ } indicate a list from which exactly one thing must be selected.

For example, in the format shown above, CMD is the command name and would appear exactly as shown. Operand1 is required and some appropriate coding must be supplied by the user. The rest of the operands are optional; however if the second is coded it must be coded as 1 or 2. If the second is omitted and the third is supplied the commas between them must be included. The left and right parentheses around the operand list must be coded as shown. The section labeled WHERE will give amplifying information concerning the types of operands, etc.

The individual commands are given in the Appendices with a detailed description of their format and usage.

V. SYSTEM COMMAND CAPABILITIES

Three levels of user interaction with DEBUG have been previously specified, and to each of these levels various commands are applicable. There is no need for the user to explicitly state which level of interaction he intends to use, rather the level is implied by the user's requests and actions. A discussion of the command features available within each level follows. All commands at any level are also available for use within the levels above them. The details and format of each command appear in the appendices.

A. NON-INTERFERENCE LEVEL

In the non-interference mode of operation, it is possible to operate the DEBUG package without alerting the XDS 9300 that it is being used. In fact, this mode is well suited to attempting to diagnose a program failure after it occurs. The debugging system operates as an observer and interpreter. The following commands are available:

1. DUMP Command

The DUMP command allows the user to display the contents of eight to one hundred sixty locations in the XDS 9300 core memory on the AGT-10 display screen. The display format is that of the familiar octal and BCD memory dump, eight locations per line, with the addresses being dumped listed along the left margin. The dump is dynamic and allows the user to view changes in the selected core section as they occur.

2. UNDUMP Command

The UNDUMP command terminates operation of a previously initiated DUMP command.

3. HOLD Command

The HOLD command allows the user to disable the dynamic feature of the current DUMP display, that is, the display will not be updated to reflect changes that may occur in the area of XDS 9300 core memory being displayed.

4. FREE Command

This command is the inverse of the HOLD command, and causes dynamic updating of the DUMP display to recommence.

5. PAGE Command

The user may cause the section of core being displayed to change without rerequesting a DUMP by the use of this and the following command. PAGE changes the display to DUMP the section of core immediately following the one currently being displayed. The length of the section displayed remains the same. If the DEBUG system is currently in the HOLD mode, the display will not be altered but all PAGE commands will be stored for cumulative operation when the display is FREEd.

6. BACK Command

This command functions as the inverse of the PAGE command, and causes the display to present the immediately preceeding portion of

core memory. It also operates under the influence of the HOLD and FREE commands.

7. LIST Command

The LIST command allows the user to display selected individual XDS 9300 core memory locations, which need not be contiguous. The display is presented dynamically with one location per line, giving the symbolic name if any, the octal address of the location and the representation of its contents in octal, decimal, floating point and BCD.

8. UNLIST Command

The UNLIST command selectively removes an individual location from the set of locations being LISTed, or may be used to delete the entire LIST display.

9. GATED Command

This command presents a display of the status words for the graphics and text interface between the other AGT-10 and the XDS 9300. Included are a description of the mode indicated by the status word, the graphics or text block referenced and an octal dump of a portion of the block if graphics, or an octal and BCD dump of the entire block if text. The display is dynamic in nature, similar to the DUMP and LIST commands.

10. Symbol Definition Command

This command allows the user to equate values with various symbols for later use as command operands. It is primarily intended as a typing and memory convenience for the user, in that once a name

is associated with a particular address or value the user may type that name rather than the octal, decimal or BCD representation of the address or value.

11. Value Inquiry Command

The value inquiry command allows the user to determine the current value of a symbol, if defined, or to evaluate a simple arithmetic expression of operands of varying type. When examining a dump it is particularly useful for performing base eight address relocation.

12. DELETE Command

The DELETE command removes a single symbol from the symbol table, or at the user's option, will clear the entire table.

13. AMRMX Command

This command causes the DEBUG system to immediately return control to the AMRMX monitor system. If DEBUG is subsequently reentered without being reloaded, it will recommence from the status at which the monitor return occured.

B. INTERACTIVE LEVEL

At the interactive level, the user may cause modification of the execution-time copy of his program or interrupt the course of program execution by use of several commands:

1. FILL Command

The FILL command allows the user to replace the contents of one to one hundred sixty contiguous cells of XDS 9300 core memory

with a value specified in octal, decimal, floating-point or BCD. The operation of FILL is immediate and non-recurring, and typically used for altering the values of higher-level language program variables.

2. BABY Command

This command allows the user to commence a babysitting operation on a given core memory location in the XDS 9300. It causes the generation of a LIST of the given location, and initiates a continuing check on the value of the contents of the location. When the value of the location's contents pass outside the range specified by the user, the XDS 9300 is halted and the appropriate LIST entry in the display is flagged.

3. TRAP Command

This command places a breakpoint trap at the specified location. When the location is subsequently executed by the XDS 9300 program, its execution is suspended and a message appears on the AGT-10 display screen providing the location of the trap and the contents of various XDS 9300 registers.

4. UNTRAP Command

This command removes a previously placed breakpoint trap from the specified location.

5. ATEX Command

This command causes execution of the user's XDS 9300 program to be suspended as execution is commenced. It must be issued prior to the execution phase of the user's program, such as during compilation or loading.

6. STOP Command

STOP causes the XDS 9300 program execution to be suspended.

7. GO Command

The GO command restarts XDS 9300 execution of the user's program at the location from which it was suspended, or at another user specified location. This command is used to recover from a BABY, TRAP, ATEX, or STOP command.

8. SLOW Command

The function of this command is to allow the user to specify a slower rate of execution for a program on the attached computer. Execution speed may be decreased by any ratio in the range of one tenth to one hundred-thousandth the normal execution speed. A continuous display of the contents of the location, A, B, and index registers of the XDS 9300 is automatically provided on the AGT-10 graphics display while in the SLOW mode.

9. FAST Command

This command restores the XDS 9300 to full execution speed and removes the dynamic display of the XDS 9300 registers.

C. AUTOMATIC SYMBOL LEVEL

At the highest level of interaction with the user's program, it is necessary that the program be compiled and executed with the foreknowledge that debugging with the use of DEBUG will be expected. However, with this foreknowledge, the programmer may have the

obvious advantage of letting DEBUG predefine the symbols that occur within his higher-level language program, thus facilitating his reference to them while debugging. This feature is enabled within DEBUG with the use of the SYMBOLS command:

1. SYMBOLS Command

The SYMBOLS command will transfer information from the XDS 9300 program concerning the names of program variables and entry points into the DEBUG system. It may only be executed once execution of the user's program on the XDS 9300 has commenced, and requires that a NAMELIST statement be included in each FORTRAN program for which the symbols are to be defined. It is convenient to use the ATEX command to hold execution so that SYMBOLS may be executed.

VI. INTERNAL SYSTEM STRUCTURE

Sufficient information appears elsewhere in this paper to assist the user in operating the DEBUG system. The material in this section is intended primarily for those concerned with internal structure as might be the case if the system were extended or modified for use at another installation.

A. INTERRUPT HANDLING CONSIDERATIONS

A primary consideration in the programming of the system is the extensive use of the interrupt structure of the AGT-10. All input-output on the teletypewriter, the channel to the XDS 9300, and the graphics display subsystem operates on the basis of hardware interrupts. It is important to note that as the graphics display is continually in use, and generates an interrupt for each word of data to be fetched for display, interrupts may occur at any point in the programming. Delays in processing these interrupts may cause unusual display images, or at best an extremely annoying display flicker. A secondary consideration here is that a display buffer may frequently be in the process of being referenced for display at the same time as it is being filled. This requires that program code must maintain a legal display representation in the display buffer, i.e. it may not be used for temporary storage, it must be filled in a strictly forward manner, and a valid terminator must always appear following the buffer contents.

B. COMMAND PROCESSING

The major portion of the DEBUG system is devoted to processing of user commands. Indeed, the order of program segment execution within the system is dependent almost entirely upon the operator. Command processing may be thought of as a series of subroutines specifically initiated either immediately or under control of the main commutator. The interface between these subroutines and the user is the command decoder.

1. Command Decoder

The command decoder is entered from a gate set in the main commutator upon receipt of the carriage return character from the teletypewriter. Thus, command decoding and execution do not occur at interrupt time, but rather at the next circuit around the commutator loop.

The decoder is a context dependent scanner based upon a predefined table of delimiters. The occurrence of any delimiter denotes the end of an argument, which includes all characters typed since the last delimiter. The defined delimiters include the following characters:

carriage return =) (, + - .

Of these, the last three characters are pseudo delimiters, and denote the end of pseudo arguments. A pseudo argument is one that is not complete in itself, such as a subfield of an arithmetic expression. All pseudo arguments between normal delimiters are

collected, combined in the appropriate manner, and stored as a single argument. For each argument information is stored as to the value, type, length and trailing delimiter. Up to six arguments may be processed. Octal and decimal arguments are converted to the appropriate numerical values. Alphameric arguments are checked against the symbol table, and replaced by the value attached to the symbol if defined. Otherwise, such arguments are stored in a BCD representation with four characters per word, right justified with a leading BCD zero. This format corresponds directly to the internal XDS 9300 representation and its shorter 24 bit word length.

After the command has been scanned, and if no errors have been detected, the command is interpreted and executed. If the first delimiter is an equals sign it is presumed to be a symbol definition or value inquiry command and is processed accordingly. A first delimiter of a left parenthesis or carriage return is the only other acceptable circumstance and implies that the first argument is a command name. This command name is looked up in a table of defined commands, and the corresponding subroutine call to a command processor is executed.

2. Command Processors

Each of the command processors is an independent subroutine. These subroutines either perform the requested action or set appropriate gates in the commutator for later execution as in the case of DUMP, LIST, BABY, TRAP and similar commands of a recurring or continuous nature. In either case, all necessary information is

extracted from the command decoder argument table, so that further input may be received. This information is stored in specific variables, such as the DUMP starting address and length, or in tables, as is the case with LIST to record the locations being displayed. The LIST table provides entries for minimum and maximum allowable values of the associated location and thus functions as the reference storage for BABY.

A considerable portion of the command processors is devoted to the conversion of data from one format to another, which is required due to the differences between the XDS 9300, AGT-10, teletypewriter and graphics display subsystem representations of symbolic and numerical values. This is provided in part by a set of conversion subroutines in the input-output section for those requirements that are of a general and repetitive nature. Due to the frequency with which some program segments are executed, such portions have been coded for maximum speed of execution with less regard to the amount of memory space required for the coding.

C. SYMBOL TABLE ORGANIZATION

The need for an ability to define frequently used symbols is obvious for an interactive debugging system such as DEBUG. However, due to the nature of the system the references to the table are fairly infrequent, and access time is not a major concern. For greatest user convenience, the table must be accessible using either the symbol or its value as the key to retrieve the other entity. This enables the system to readily

attach all defined symbols to portions of XDS 9300 core memory appearing on the display in commands such as LIST. Furthermore, it is desirable to arrange the table so that it is searched in a well-defined order, thus facilitating the use of qualified symbol names. If the user refers to a qualified symbol without the qualifier, he must be able to predict which reference will be extracted. Due to these considerations, the symbol table was implemented as a simple list, and is searched bottom-up by either of two subroutines for referencing the symbol or its value. The symbol table is designed to overlay the AMRMX monitor if it extends beyond its reserved area, and thus has room for at least 1500 symbols.

D. XDS 9300 COMMUNICATION

Communication between DEBUG and the XDS 9300 occurs on two levels, which correspond essentially to the two lowest levels of DEBUG interaction.

1. Non-Interactive Communication

In this mode DEBUG simply observes the contents of selected portions of XDS 9300 core memory. This is accomplished with a subroutine to drive the Adage Interface Multiplexer and Memory Interface which attaches to the XDS Memory Interface Channel. The subroutine can transfer any length of block from any location in either machine to the other. In use, the data from the XDS 9300 is transferred to reserved areas in DEBUG depending upon the command requesting the transfer.

2. Interactive Communication

For commands requiring control of XDS 9300 program execution, it is necessary to enable the interrupt system between the computers with the AGT control card under the XDS Real Time Monitor. DEBUG then inserts a short interrupt processing and control routine in the AGT tape buffer area in upper XDS 9300 core memory. Control is maintained and exercised with the use of interrupts in both directions initiated by DEBUG and the coding it has inserted in the XDS 9300 and by appropriate alterations in the contents of the XDS 9300 core memory contents using the subroutine previously described.

VII. USING THE DEBUGGING SYSTEM

Utilization of any debugging system is highly dependent upon the nature of the problems which the user is attempting to trace. DEBUG is intended particularly as an aid for debugging execution-time errors in FORTRAN and assembly language programs on the XDS 9300. Although it might certainly be useful in detecting, locating and eliminating bugs under other circumstances on this hardware configuration, this paper will be restricted to describing the debugging process for the aforementioned intended usage.

In the case of a program which is known to contain errors, DEBUG would normally be initiated prior to the execution time phase of the user's program. This allows the user to hold execution by the use of the ATEX command, which provides a convenient time to prepare for the debugging to follow. The user should have previously considered whether or not he wishes to use the automatic symbol definition feature, and if so he would normally request the SYMBOLS command at this time. Otherwise, using the information from the compiler and assembler reference tables and loader memory allocation maps printed by the XDS 9300 monitor system prior to execution, the user might define pertinent symbols with the symbol definition command. A convenient method of doing this is to first define the program origin of the entire user segment, then define the individual program subroutine names relative to the

segment according to the relative segment map, and finally define specific program variable names relative to their respective subroutine names. Such relative addressing is easily performed by the use of the simple octal arithmetic addition expression in operands such as X=PROG1+537.

After providing for the definition of such symbols as may be convenient, the user would frequently set a decreased rate of execution, LIST or DUMP appropriate variables or program portions and allow execution to proceed. The reduced execution rate, obtained by the use of the SLOW command, allows a user to more readily observe the progress of his program, as well as providing continuously updated copies of the XDS 9300 program referenceable machine registers on the AGT-10 display screen. One of these, the location register, provides the user with the location of the current instruction being executed, thus facilitating the attempt to follow program flow.

If the user becomes aware that his program is destroying portions of code or data he might examine a DUMP of selected portions and reexecute the program with babysitting enabled on suspected index variables. This provides an easy means of detecting array references beyond the bounds of the array if the babysitting bounds are appropriately chosen. Errors in program logic may frequently be detected with judicious use of the LIST, BABY, and TRAP commands. TRAP is often used as a substitute for the numerous WRITE or PRINT statements inserted in program decks by novice programmers attempting

to trace program flow. By TRAPing and using a LIST or DUMP of appropriate variables when the trap occurs, similar information may be obtained entirely at the symbolic level on a faster, interactive basis.

During the course of debugging user programs, it is often found that errors in programming may be temporarily patched for testing purposes online without the need to recompile the program. For instance, one of the frequent results of keypunch errors is that a program may refer to the wrong variable at some point in the program. If the user is familiar with the XDS 9300 machine language, he might alter the reference to use the correct variable using FILL. Otherwise, the user may alter the value of the intended variable as necessary during the course of execution using FILL to do the altering and using TRAP to alert him to the need for an alter.

DEBUG also provides a convenient means for testing a working program with varying data or parameter values to optimize some aspect of program performance. A TRAP command might be inserted at a point near the termination of the program. Each time the trap occurs, the user may alter the desired parameter values using the FILL command, and restart the program near the beginning with the GO command. It should be noted that it is possible for the user to define FORTRAN statement numbers (including at least one non-numeric character, such as 10S) from the information printed at compilation and loading in a similar manner to defining program variable names.

EXAMPLE

The following command causes XDS 9300 program execution to proceed at one-hundredth the normal rate:

```
SLOW(100.)
```

COMMAND DESCRIPTION

COMMAND	OPERANDS
FAST	

FUNCTION

This command functions to restore the normal rate of XDS 9300 program execution. No operands are necessary. An error message is generated if DEBUG is not in the SLOW mode.

COMMAND DESCRIPTION

COMMAND	OPERANDS
SYMBOLS	

FUNCTION

This command causes automatic definition in the DEBUG symbol table of all defined entry points and NAMELIST variable names in the XDS 9300 program. It must be executed only after the execution phase of the user's program has commenced.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	LCDR E. A. Singer, (Code 53Sf) Department of Mathematics Naval Postgraduate School Monterey, California 93940	1
4.	ENS Allan Warren Walker, SC, USN Naval Supply Corps School Athens, Georgia	1
5.	Electrical Engineering Department Hybrid Computer Laboratory, Code 52EC Naval Postgraduate School Monterey, California 93940	3

DOCUMENT CONTROL DATA - R & D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1 ORIGINATING ACTIVITY (Corporate author)
Naval Postgraduate School
Monterey, California 93940

2a. REPORT SECURITY CLASSIFICATION
Unclassified

2b. GROUP

3 REPORT TITLE

An Interactive Graphical Debugging System

4 DESCRIPTIVE NOTES (Type of report and, inclusive dates)
Master's Thesis; June 1971

5 AUTHOR(S) (First name, middle initial, last name)

Allan W. Walker

6. REPORT DATE
June 1971

7a. TOTAL NO. OF PAGES
63

7b. NO. OF REFS

8a. CONTRACT OR GRANT NO.

b. PROJECT NO.

c.

d.

9a. ORIGINATOR'S REPORT NUMBER(S)

9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10. DISTRIBUTION STATEMENT

Approved for public release; distribution unlimited.

11. SUPPLEMENTARY NOTES

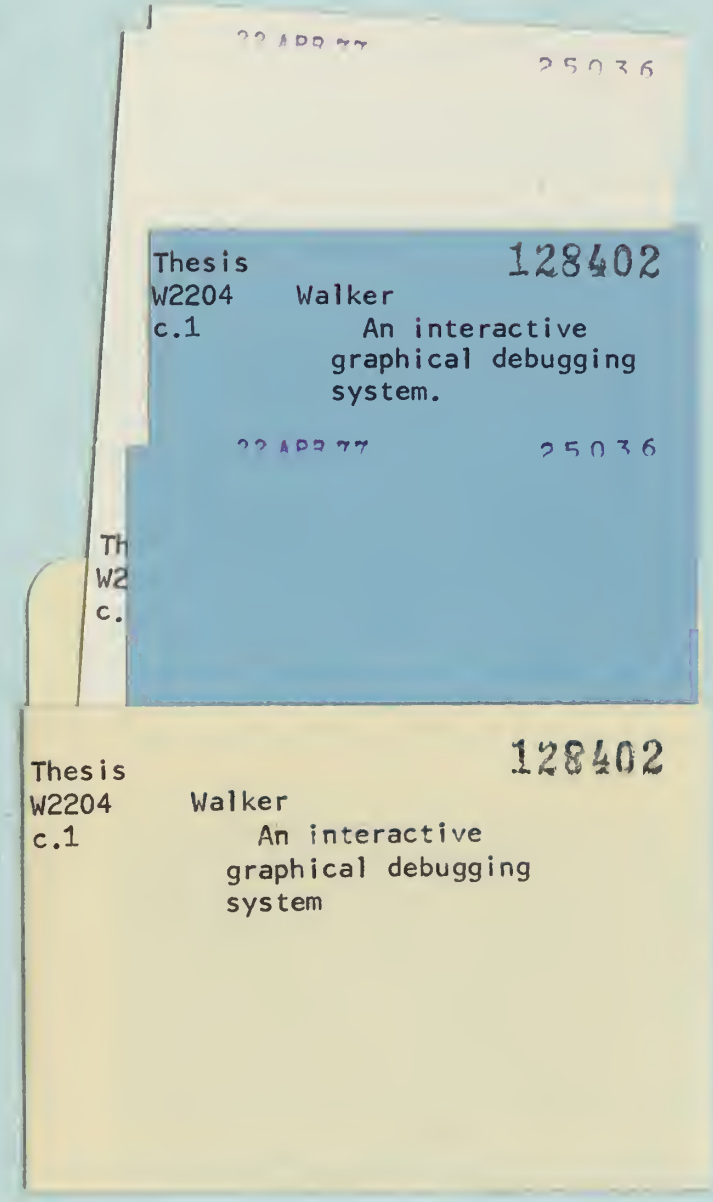
12. SPONSORING MILITARY ACTIVITY
Naval Postgraduate School
Monterey, California 93940

13. ABSTRACT

A system is described which provides an interactive graphical debugging facility for user programs. This system is implemented on an Adage AGT-10 and is operational for online debugging of higher-level language programs executing on an XDS 9300 host computer. System architecture and implementation are discussed. A formal definition of the DEBUG Command Language is given and a description of the utilization of the commands for program debugging is presented.

DD FORM 1 NOV 65 1473 (PAGE 1)
S/N 0101-807-6811

KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
Debugging						
Interactive Graphics						
Online Debugging						
Program Checkout						


22 APR 77 25036

Thesis 128402
W2204 Walker
c.1 An interactive
graphical debugging
system.

22 APR 77 25036

Th
W2
c.

Thesis 128402
W2204 Walker
c.1 An interactive
graphical debugging
system

thesW2204
An interactive graphical debugging syste
3 2768 001 92859 1
DUDLEY KNOX LIBRARY